Truck Platooning: week 8

Changyao Zhou, Jiaxin Pan

February 15, 2022

1 Introduction of the project

The goal of this project is to chase a chosen target vehicle by predicting the steering angle and throttle input of the ego-vehicle, given the velocity of both vehicles and images taken from cameras.

The control values (steering angle and throttle) computed by Model Predictive Control (MPC) are used as ground-truth. MPC is an advanced method of process control that is used to control a process while satisfying a set of constraints[WIK]. It takes a prediction horizon instead of a single time step into account, and aims to get an optimal control result minimizing the objective function within the prediction horizon. For learning the MPC controller, we take [MPC] as reference.

The project is designed to be realized in the following steps:

1). Control the ego-vehicle with Model Predictive Control (MPC) in the Carla simulator

2). Given x, y-location, yaw angle and velocity of both vehicles, train a fully connected neural network to control ego-vehicle by using the MPC output as ground-truth.

2). Given object velocity, ego-velocity and RGB image, train a neural network to control the ego-vehicle by using the MPC output as ground-truth

3). Further improvement by changing the input of the neural network

2 Overview of this week

Last week we successfully got a good result of MLP model, which has Δx, Δy, Δyaw of relative transformation between two vehicles as input. We tried the model inference both on the known trajectories in the dataset and the unknown trajectories in Town04 and other towns. At most time, the ego-vehicle can stably follow the target vehicle.

This week we continued to train and improve CNN model with image and velocity input. This time we tried three approaches, CNN-MLP, CNN-MPC, end-to-end CNN. And the results of all three approaches are generally acceptable. In addition, we tried to do image rendering from the off-trajectory points to see if it can be used for data collection.

All results of models mentioned above were tested with the 'online' mode in CARLA, where the target vehicle is set in 'autopilot' mode. All the videos are stored under https://syncandshare.lrz.de/getlink/fiKDDyecmv5wtrmuQFYiMA2d/week8.

3 Training of CNN Model

3.1 CNN-MLP Model

Last week we tried to train an end-to-end CNN model that has images and two velocities as input and directly predict throttles and steering angles as output. The validation loss remain high and the ego-vehicle cannot follow the target vehicle stably. This time, we want to divide the whole process into two steps. One is to use images from ego-vehicle and velocities to predict the relative transformation. Next step is to feed the predicted relative transformation, Δx, Δy, Δyaw, as well as two velocities into our trained MLP and predict the two control values, throttle and steering angle.

Table 1: CNN Architecture

Layer Number	Layer Type	Layer Input	Layer Output
1 - 13	feature extractor from pretrained Alexnet	3x128x128	256x3x3
14	MaxPooling	256x3x3	256x1x1
15	Flatten	256x1x1	258
16	Linear Layer	258	64
17	ReLU	64	64
18	Linear Layer	64	3

Through the training, we found the pretrained Alexnet architecture has better performance on extracting relative transformations from the images. The detailed architecture is shown in Table 1. After training for around 100 epoches, the train loss has been reduced to 5e-3, the validation loss has reached 1.8e-2. The losses for three prediction (Δx, Δy, Δyaw) are 4e-2, 1e-2, 1e-3 respectively. The inference of relative transformation is relatively accurate. Then we fed the prediction and velocities into the MLP from last week. The result of car following in CARLA seems good.

We used CNN-MLP model to do inference on different trajectories in Town04 and Town03. As can be seen from the videos, the model seems to work well with sharp turns in Town 04. And in Town 03 there is a round turn in another town, which requires very precise steering control values, and the ego vehicle can follow the target vehicle in a reasonable way. However, one of the problem occurs at some sharp turn in Town 03. When turning, only a small part of the target vehicle may appear in the view of the ego-vehicle, which is not conducive to it behind judging its relative position to the target vehicle. In this case the ego-vehicle would lose the position of target and can't follow any more.

All the videos are stored under https://syncandshare.lrz.de/getlink/fiVuEwNR1pgfwc3EgAgDGWrp/CNN_3output_MLP.

3.2 CNN-MPC Model

Similar to the case above, we use the same image extractor model to predict relative transformation between the ego-vehicle and target vehicle. Now we want to feed the states of both vehicles into the MPC controller to get more accurate control values. With known relative transformation from the target vehicle to ego-vehicle, we set the absolute transformation of target vehicle as I_{3x3} with x = 0, y = 0, yaw angle = 0. Then we have

$$G_2^0 = T_2^1 G_1^0 = T_2^1 I_{3x3} = T_2^1 \tag{1}$$

$$= \begin{bmatrix} cos(\Delta\theta) & -sin(\Delta\theta) & \Delta\text{x} \\ sin(\Delta\theta) & cos(\Delta\theta) & \Delta\text{y} \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

where G_1^0 is absolute transformation of target vehicle, G_2^0 is absolute transformation of ego-vehicle, T_2^1 is relative transformation from target vehicle to ego-vehicle. Δx, Δy and $\Delta\ \theta$ are predicted relative difference of position and yaw angle between two vehicles.

Then we can feed the states of both vehicles, including x-, y-position, yaw angle, and velocity into MPC controller. Considering minimizing objective function within predict horizon, MPC can output the optimal control values according to current states of both vehicles.

The inference result has no obvious improvement or difference compared with the CNN-MLP approach above. The ego-vehicle can generally follow the target vehicle stably without collision. All the videos are stored under https://syncandshare.lrz.de/getlink/fiJasQwWQvKqrxwgxgaQTfWi/CNN_MPC.

3.3 End-to-End CNN Model

The end-to-end CNN model is similar to the one we used to train last week. But this time we added three more outputs. That means the CNN model would predict not only the two control values, throttle and steering angle, but also three delta value for relative transformation. The reason why we add these

three predictions is that we want to help the model to first predict the relative transformation, and then predict control values based on it. It might probably have a more accurate prediction.

With this end-to-end CNN model, we got an average train loss 1.5e-3, an average validation loss 3.6e-2. The losses for five outputs (Δx, Δy, Δyaw, throttle, steering angle) are 8e-2, 6e-2, 1e-3, 3e-2, 2e-2 respectively. Since the validation loss of the two-output CNN model is around 2e-2, we can see that after adding three more outputs, the losses of the two control values didn't actually significantly reduced. However, the inference shows that the CNN model can make accurate predictions for relative transformation. That also tells us that it's easier for CNN to extract relative transformation information than to predict control values.

In addition, it's worth to notice that last week we set the camera position incorrectly when doing inference. This is possible reason of the shaking of ego-vehicle. This week, we have fixed the camera position and got a better performance. The ego-vehicle can generally follow the target vehicle stably without collision. All the videos are stored under https://syncandshare.lrz.de/getlink/fi8zTeG2NrePRG34pSuHKSv7/CNN.

4 Image Rendering for Data Collection

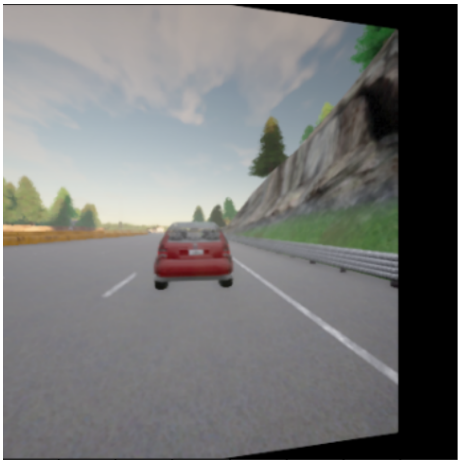

Figure 1: Rotation 8°

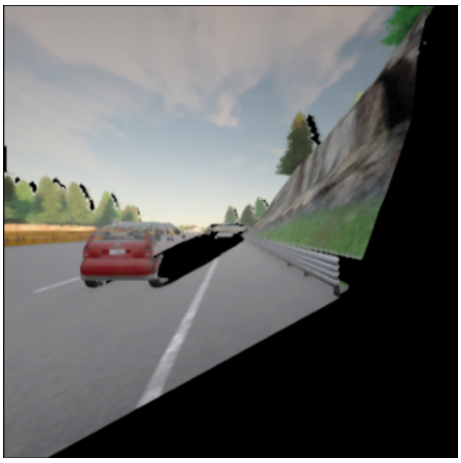

Figure 2: Lateral Offset 2m

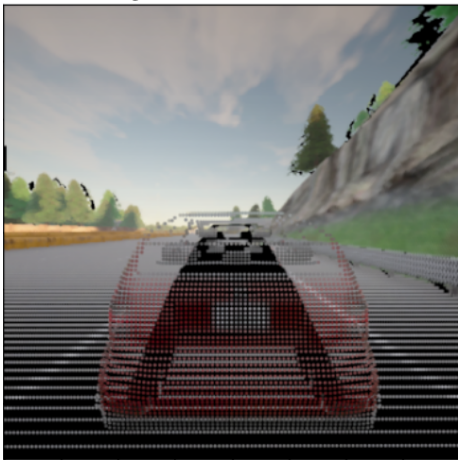

Figure 3: Longitudinal Offset +3m

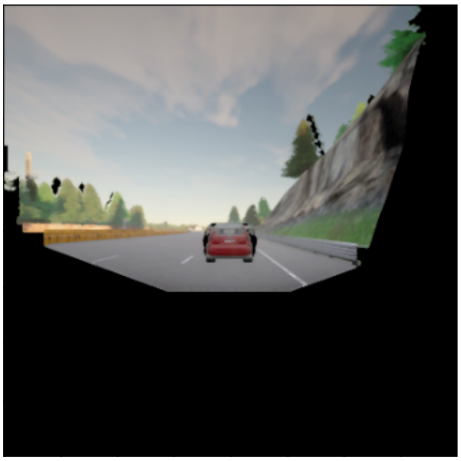

Figure 4: Longitudinal Offset -3m

When we collect the data before, we added a random offset to the ego-vehicle in order to collect the off-trajectory data. However, this is not possible in real-world data collection. To solve this problem, we can use the similar approach for image rendering as exercise in week4. This time we only need to collect the on-trajectory samples. Then each time we randomly sample a point around the trajectory,

with known position of the ego-vehicle, we can do image rendering to get the view of ego-vehicle, and use MPC controller to compute the ground truth control values. And the whole process of collecting off-trajectory samples can be done without CARLA simulator.

However, there are several problems for this approach. First, The acceptable range of random offsets for this image rendering approach is very small. To generate a complete image without large black holes, we need to ensure the view of original image must include the view after random transformation. Otherwise, missing parts in the image cannot be filled by rendering. In the figures above, we can see several examples of rendering failure due to too much deflection. Through our attempts, we found that only slight transformations are allowed, namely positive longitudinal offset less than 1m (moving forward), lateral offset within ± 0.1m, rotation within $\pm 5°$. This is much smaller than the offset range of our data collection before.

In addition, the image rendering approach has the similar problem as the offline mode for data collection before. We can't get the accurate plant model to update the states of both vehicles with control values. With inaccurate states, we can't get accurate ground truth of the following control values from MPC controller. And a third problem is that the image rendering process is time-consuming. As a result, off-trajectory data collection using image rendering is not a good idea for our case.

5 Conclusion and future work

In this week, we tried three approaches based on CNN model, CNN-MLP, CNN-MPC, end-to-end CNN. The first two approaches have the similar result, where the ego-vehicle can generally follow the target vehicle. But sometimes it will miss the target in the view at sharp turn and get lost. And for the third approach, we don't have big improvements compared to last week. But the ego-vehicle can also follow the target at most time. In addition, we tried the image rendering for data collection. It turns out that it can actually work for off-trajectory data, but it's not suitable for us because of the small range and long processing time.

For the next step, we can start to discard the velocities and use only images as input. Perhaps RNN will be used to predict the velocities of both vehicles with a consecutive sequence of images. If the RNN can work well, then we can implement the automatic car following only with images, which is similar with the real situation, where we follow the target vehicle only based on the view while driving.

References

[MPC] Model Predictive Control. https://github.com/WuStangDan/mpc-course-assignments.

[WIK] Model Predictive Control. https://en.wikipedia.org/wiki/Model_predictive_control.